

A stylized, white, 3D-rendered city skyline with various skyscrapers and buildings, set against a light gray background with a subtle diagonal line pattern. The skyline is centered horizontally and partially obscured by a dark blue banner.

PySpark Basic

讲师：轩宇



PySpark

Lesson Plan

Installing Python Linux

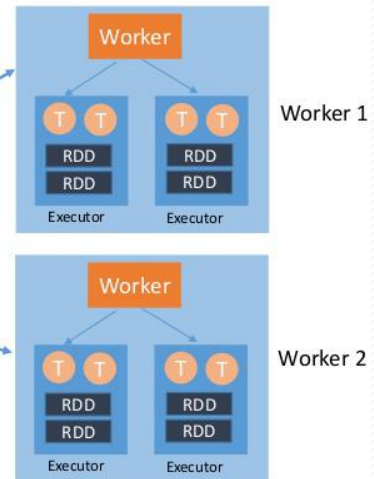
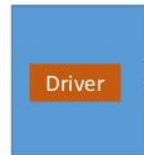
Spark Deployment

Submit Application

User Action Analyser

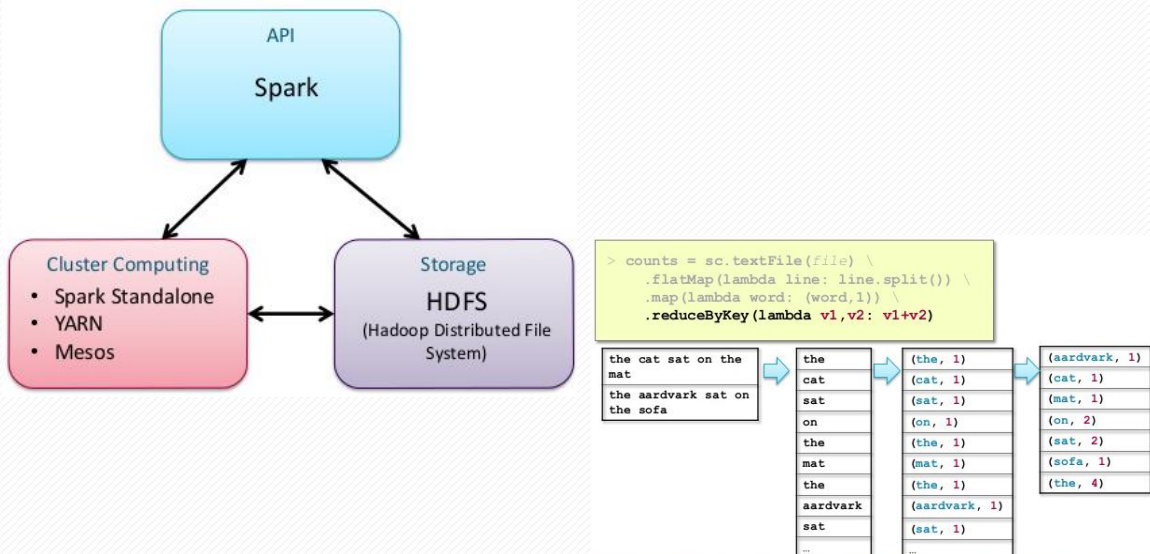
SPARK ARCHITECTURE

- **Driver**
 - Driver program that runs the user's main function and executes various parallel operations on a cluster
- **RDDs**
 - Collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.



- **Worker**
 - Manages resources on cluster node
- **Executor**
 - Is the JVM process which stores and executes tasks
- **Tasks**
 - Executes the RDD operations

DISTRIBUTED PROCESSION WITH SPAK



SPARK V. HADOOP MAPREDUCE

- Spark takes the concepts of MapReduce to the next level
 - Higher level API = faster, easier development



```
> counts = sc.textFile(file) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1)) \
    .reduceByKey(lambda v1,v2: v1+v2)
> counts.saveAsTextFile(output)
```



```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Job job = new Job();
        job.setJarByClass(WordCount.class);
        job.setJobName("Word Count");
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(WordMapper.class);
        job.setReducerClass(SumReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}

public class WordMapper extends Mapper {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        for (String word : line.split("\\W+")) {
            if (word.length() > 0)
                context.write(new Text(word), new IntWritable(1));
        }
    }
}

public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int wordCount = 0;
        for (IntWritable value : values) {
            wordCount += value.get();
        }
        context.write(key, new IntWritable(wordCount));
    }
}
```

PYSPARK BASICS

Python For Data Science Cheat Sheet

PySpark Basics

Learn Python for data science <https://www.datacamp.com>

Spark
PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master="local[2]", appName="PySpark")
```

Inspect SparkContext

```
>>> sc.version() # Return SparkContext version
>>> sc.defaultParallelism() # Return default parallelism
>>> sc.getConf() # Return SparkContext configuration
>>> sc.getConf().get("spark.master") # Return SparkContext master URL
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions() # List the number of partitions
>>> rdd.count() # Count RDD instances
>>> rdd.getNumPartitionsByKey() # Count RDD instances by key
>>> rdd.getNumPartitionsByValue() # Count RDD instances by value
>>> rdd.getNumPartitionsByMap() # Count RDD instances by map
>>> rdd.getNumPartitionsBySum() # Sum of RDD elements
>>> rdd.getNumPartitionsByCheck() # Check whether RDD is empty
```

Summary

```
>>> rdd.max() # Maximum value of RDD elements
>>> rdd.min() # Minimum value of RDD elements
>>> rdd.mean() # Mean value of RDD elements
>>> rdd.stdev() # Standard deviation of RDD elements
>>> rdd.var() # Variance of RDD elements
>>> rdd.histogram() # Compute histogram of RDD elements
```

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y: x+y) # Merge the RDD values for each key
>>> rdd.reduce(lambda x,y: x+y) # Merge the RDD values
```

Coalescing

```
>>> rdd.coalesce(n) # Return RDD of grouped values
>>> rdd.coalesce(n, true) # Return RDD of grouped values, shuffle
```

Aggregating

```
>>> rdd.aggregate(0, lambda x,y: (x,y), lambda x,y: (x,y)) # Aggregate RDD elements of each partition and then the results
>>> rdd.aggregate(0, lambda x,y: (x,y), lambda x,y: (x,y)) # Aggregate the elements of each partition, and then the results
```

Check Out PySpark Cheat Sheet

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
>>> sc.help() # Show the help of the SparkContext
>>> sc.help("sc") # Show the help of the SparkContext
```

Set which master the context connects to with the `--master` argument, and add Python `db` option to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([1,2,3,4,5])
>>> rdd = sc.parallelize([1,2,3,4,5], numSlices=4)
>>> rdd = sc.parallelizeRange(100)
>>> rdd = sc.parallelizeRange(100, 200, numSlices=4)
```

External Data

```
>>> sc.textFile("hdfs://...") # Read text files from HDFS
>>> sc.textFile("file:///...") # Read text files from local file system
>>> sc.textFile("jdbc://...") # Read text files from JDBC
```

Selecting Data

Getting

```
>>> rdd.collect() # Return a list with all RDD elements
>>> rdd.take(n) # Take first n RDD elements
>>> rdd.first() # Take first RDD element
>>> rdd.top(n) # Take top n RDD elements
```

Sampling

```
>>> rdd.sample(withReplacement=True, fraction=0.1, numPartitions=10) # Return sampled subset of RDD
```

Filtering

```
>>> rdd.filter(lambda x: x % 2 == 0) # Filter the RDD
>>> rdd.filter(lambda x: x % 2 != 0) # Filter the RDD
>>> rdd.filter(lambda x: x % 2 == 0) # Filter the RDD
```

Iterating

```
>>> rdd.foreach(lambda x: print(x)) # Apply a function to all RDD elements
>>> rdd.foreachPartition(lambda iter: iter.close()) # Apply a function to all RDD elements
```

Manipulating Data

Map

```
>>> rdd.map(lambda x: x*2) # Map RDD by given function
>>> rdd.mapPartitions(lambda iter: iter.close()) # Map RDD by given function
```

MapPartitions

```
>>> rdd.mapPartitions(lambda iter: iter.close()) # Map RDD by given function
>>> rdd.mapPartitions(lambda iter: iter.close()) # Map RDD by given function
```

MapPartitionsWithIndex

```
>>> rdd.mapPartitionsWithIndex(lambda iter, index: iter.close()) # Map RDD by given function
>>> rdd.mapPartitionsWithIndex(lambda iter, index: iter.close()) # Map RDD by given function
```

MapPartitionsWithKey

```
>>> rdd.mapPartitionsWithKey(lambda iter, key: iter.close()) # Map RDD by given function
>>> rdd.mapPartitionsWithKey(lambda iter, key: iter.close()) # Map RDD by given function
```

MapPartitionsWithKeyAndIndex

```
>>> rdd.mapPartitionsWithKeyAndIndex(lambda iter, key, index: iter.close()) # Map RDD by given function
>>> rdd.mapPartitionsWithKeyAndIndex(lambda iter, key, index: iter.close()) # Map RDD by given function
```

Partitioning

Repartitioning

```
>>> rdd.repartition(n) # New RDD with n partitions
>>> rdd.repartition(n, true) # New RDD with n partitions, shuffle
```

Coalesce

```
>>> rdd.coalesce(n) # Return RDD of grouped values
>>> rdd.coalesce(n, true) # Return RDD of grouped values, shuffle
```

Saving

Save

```
>>> rdd.saveAsTextFile("hdfs://...") # Save RDD as text file
>>> rdd.saveAsTextFile("hdfs://...") # Save RDD as text file
```

SaveAsSequenceFile

```
>>> rdd.saveAsSequenceFile("hdfs://...") # Save RDD as sequence file
>>> rdd.saveAsSequenceFile("hdfs://...") # Save RDD as sequence file
```

SaveAsHadoopFile

```
>>> rdd.saveAsHadoopFile("hdfs://...", formatClass="org.apache.hadoop.mapred.SequenceFileWriter") # Save RDD as Hadoop file
>>> rdd.saveAsHadoopFile("hdfs://...", formatClass="org.apache.hadoop.mapred.SequenceFileWriter") # Save RDD as Hadoop file
```

Stopping SparkContext

```
>>> sc.stop() # Stop SparkContext
```

Execution

```
>>> sc.executeCommand("echo hello") # Execute command
>>> sc.executeCommand("echo hello") # Execute command
```